Amendments to the Specification:

Please amend the Specification as indicated:

At pages 1-2, please replace the section under "Cross Reference to Related Applications" as indicated:

The present invention is related to the following applications entitled "Method and Apparatus for Counting Instruction Execution and Data Accesses", serial no. 10/675,777, now U.S. Patent No. 7,395,527, issued July 1, 2008, attorney docket no. AUS920030477US1; "Method and Apparatus for Selectively Counting Instructions and Data Accesses", serial no. 10/674,604, attorney docket no. AUS920030478US1; "Method and Apparatus for Generating Interrupts Upon Execution of Marked Instructions and Upon Access to Marked Memory Locations", serial no. 10/675,831, attorney docket no. AUS920030479US1; "Method and Apparatus for Counting Data Accesses and Instruction Executions that Exceed a Threshold", serial no. 10/675,778, attorney docket no. AUS920030480US1; "Method and Apparatus for Debug Support for Individual Instructions and Memory Locations", serial no. 10/675,751, attorney docket no. AUS920030482US1; "Method and Apparatus to Autonomically Select Instructions for Selective Counting", serial no. 10/675,721, attorney docket no. AUS920030483US1; "Method and Apparatus to Autonomically Count Instruction Execution for Applications", serial no. 10/674,642, attorney docket no. AUS920030484US1; "Method and Apparatus to Autonomically Take an Exception on Specified Instructions", serial no. 10/674,606, attorney docket no. AUS920030485US1; "Method and Apparatus to Autonomically Profile Applications", serial no. 10/675,783, attorney docket no. AUS920030486US1; and "Method and Apparatus for Counting Instruction and Memory Location Ranges", serial no. 10/675,872, now, U.S. Patent No. 7,373,637, issued May 13, 2008, attorney docket no. AUS920030487US1 filed even date hereof, assigned to the same assignee, and incorporated herein by reference.

At page 15, paragraph 1:

BIU **212** is connected to an instruction cache unit **214** and to data cache unit **216** of processor **210**. Instruction cache unit **214** outputs instructions to sequencer unit **218**. In response to such instructions from instruction cache unit **214**, sequencer unit **218** selectively outputs instructions to other execution circuitry of processor **210**.

At page 16, paragraph 1:

In response to a Load instruction, LSU **228** inputs information from data cache unit **216** and copies such information to selected ones of rename buffers **234** and **238**. If such information is not stored in data cache unit **216**, then data cache unit **216** inputs (through BIU **212** and system bus **211**) such information from a system memory **239** connected to system bus **211**. Moreover, data cache unit **216** is able to output (through BIU **212** and system bus **211**) information from data cache unit **216** to system memory **239** connected to system bus **211**. In response to a Store instruction, LSU **228** inputs information from a selected one of GPRs **232** and FPRs **236** and copies such information to data cache unit **216**.

At page 16, last paragraph:

Sequencer unit **218** inputs and outputs information to and from GPRs **232** and FPRs **236**. From sequencer unit **218**, branch unit **220** inputs instructions and signals indicating a present state of processor **210**. In response to such instructions and signals, branch unit **220** outputs (to sequencer unit **218**) signals indicating suitable memory addresses storing a sequence of instructions for execution by processor **210**. In response to such signals from branch unit **220**, sequencer unit **218** inputs the indicated sequence of instructions from instruction cache unit **214**. If one or more of the sequence of instructions is not stored in instruction cache unit **214**, then instruction cache unit **214** inputs (through BIU **212** and system bus **211**) such instructions from system memory **239** connected to system bus **211**.

At page 17, paragraph 1:

In response to the instructions input from instruction cache unit **214**, sequencer unit **218** selectively dispatches the instructions to selected ones of execution units **220**, **222**, **224**, **226**, **228**, and **230**. Each execution unit executes one or more instructions of a particular class of

instructions. For example, FXUA **222** and FXUB **224** execute a first class of fixed-point mathematical operations on source operands, such as addition, subtraction, ANDing, ORing and XORing. CFXU **226** executes a second class of fixed-point operations on source operands, such as fixed-point multiplication and division. FPU **230** executes floating-point operations on source operands, such as floating-point multiplication and division.

At page 18, paragraph 2:

In the fetch stage, sequencer unit **218** selectively inputs (from instruction cache unit **214**) one or more instructions from one or more memory addresses storing the sequence of instructions discussed further hereinabove in connection with branch unit **220**, and sequencer unit **218**.

At page 20, last paragraph:

In addition, processor **210** also includes performance monitor unit **240**, which is connected to instruction cache unit **214** as well as other units in processor **210**. Operation of processor **210** can be monitored utilizing performance monitor unit **240**, which in this illustrative embodiment is a software-accessible mechanism capable of providing detailed information descriptive of the utilization of instruction execution resources and storage control. Although not illustrated in **Figure 2**, performance monitor unit **240** is coupled to each functional unit of processor **210** to permit the monitoring of all aspects of the operation of processor **210**, including, for example, reconstructing the relationship between events, identifying false triggering, identifying performance bottlenecks, monitoring pipeline stalls, monitoring idle processor cycles, determining dispatch efficiency, determining branch efficiency, determining the performance penalty of misaligned data accesses, identifying the frequency of execution of serialization instructions, identifying inhibited interrupts, and determining performance efficiency. The events of interest also may include, for example, time for instruction decode, execution of instructions, branch events, cache misses, and cache hits.

At page 21, last paragraph:

Additionally, processor **210** also includes interrupt unit **250**, which is connected to instruction cache unit **214**. Additionally, although not shown in **Figure 2**, interrupt unit **250** is connected to other functional units within processor **210**. Interrupt unit **250** may receive signals from other

functional units and initiate an action, such as starting an error handling or trap process. In these examples, interrupt unit **250** is employed to generate interrupts and exceptions that may occur during execution of a program.

At page 22, last paragraph:

Turning now to **Figure 3**, a diagram illustrating components used in processing instructions associated with indicators is depicted in accordance with a preferred embodiment of the present invention. Instruction cache unit **300** receives bundles **302**. Instruction cache unit **300** is an example of instruction cache unit **214** in **Figure 2**. A bundle is a grouping of instructions. This type of grouping of instructions is typically found in an IA-64 processor, which is available from Intel Corporation. Instruction cache unit **300** processes instructions for execution.

At page 23, paragraph 1:

As part of this processing of instructions, instruction cache unit **300** determines which instructions are associated with indicators. These indicators also are referred to as "performance indicators" in these examples. Signals **304** have been associated with performance indicators. As a result, signals **304** for the instructions are sent to performance monitor unit **306**. Performance monitor unit **306** is an example of performance monitor unit **240** in **Figure 2**.

At page 23, paragraph 2:

When instruction cache unit **300** determines that an instruction associated with an indicator is present, a signal is sent to indicate that a marked instruction is being executed. In these examples, a marked instruction is an instruction associated with a performance indicator. Alternatively, a performance indicator may indicate that all items or instructions in a bundle are marked to be counted. Additionally, signals for these instructions are sent by instruction cache unit **300** to the appropriate functional unit. Depending on the particular implementation, a functional unit other than performance monitor unit **306** may count execution of instructions. In the case that the performance indicators are in the instructions, or in the bundles, the cache unit, instruction cache unit **300**, detects the indicators and sends signals to performance monitor unit **306**.

At page 24, paragraph 1:

When signals for these instructions are received by performance monitor unit **306**, performance monitor unit **306** counts events associated with execution of instructions **304**. As illustrated, performance monitor unit **306** is programmed only to count events for instructions associated with performance indicators. In other words, an indicator associated with a instruction or memory location is used to enable counting of events associated with the instruction or memory location by performance monitor unit **306**. If an instruction is received by instruction cache unit **300** without a performance indicator, then events associated with that instruction are not counted. In summary, the performance indicators enable the counting on a per instruction or per memory location basis in a processor.

At page 24, last paragraph:

With respect to the accessing of data in memory locations, the data and indicators are processed by a data cache unit, such as data cache unit **216** in **Figure 2**, rather than by an instruction cache unit. The data cache unit sends signals indicating that marked memory locations are being accessed to performance monitor unit **306**. Marked memory locations are similar to marked instructions. These types of memory locations are ones associated with a performance indicator.

At page 28, paragraph 2:

Turning now to **Figure 7**, a flowchart of a process for processing instructions containing performance indicators is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 7** may be implemented in an instruction cache unit, such as instruction cache unit **214** in **Figure 2**.

At page 29, last paragraph:

Turning now to **Figure 8** a flowchart of a process for selectively sending signals to an interrupt unit is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 8** may be implemented in an instruction cache unit, such as instruction cache unit **242** in **Figure 2**. This process is employed in cases in which monitoring events using a performance monitor unit may miss certain events. For example, a performance monitor unit counts events. When a cache miss occurs, a signal is sent to the performance

monitor unit. When the meta data for a corresponding cache line is loaded into the cache, the appropriate signal or signals also are raised. If the meta data indicates that an exception is to be raised, then a signal is sent to the interrupt unit in which the signal indicates that an exception is to be raised.

At page 31, paragraph 1:

With reference now to **Figure 9**, a flowchart of a process for generating an interrupt in response to an access of a memory location associated with a performance indicator is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 9** may be implemented in a data cache unit, such as data cache unit ~~**246**~~ **216** in **Figure 2**.

At page 31, last paragraph:

The process begins by receiving a signal from an instruction cache unit indicating that an instruction with a performance indicator is being processed (step **1000**). Next, events associated with the instruction being processed are counted (step **1002**) with the process terminating thereafter. The counting of events may be stored in a counter, such as counter **241** in **Figure 2**.

At page 32, paragraph 2:

With reference next to **Figure 11**, a flowchart of a process for selective counting of instructions is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 11** may be implemented in an instruction cache unit, such as instruction cache unit **214** in **Figure 2**.

At page 33, paragraph 1:

The indicator in step **1100** and step **1104** may be the same indicator in which the indicator toggles the setting and unsetting of the flag. In another implementation, two different indicators may be used in which a first indicator only sets the flag. A second indicator is used to unset the flag. Communication between a cache unit, such as an instruction cache unit or a data cache unit, and the performance monitor unit to indicate a mode of counting may be implemented simply with a high signal when counting is to occur and a low signal when counting is no longer enabled.

At page 33, paragraph 2:

With reference next to **Figure 12**, a flowchart of a process for selective counting of instructions is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 12** may be implemented in an instruction cache unit, such as instruction cache unit **214** in **Figure 2**.

At page 34, paragraph 1:

Turning now to **Figure 13**, a flowchart of a process for identifying instructions exceeding a threshold is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 13** may be implemented in an instruction cache unit, such as instruction cache unit **214** in **Figure 2**.

At page 36, paragraph 2:

A similar process may be implemented in a data cache unit, such as data cache unit **216** in **Figure 2** to monitor accesses to memory locations. The process illustrated in **Figure 13** may be adapted to identify the cycles needed to access data in a memory location. As with the execution of instructions, counting occurs or an interrupt is generated when the amount of time needed to access the data in a memory location exceeds a specified threshold.

At page 36, last paragraph:

With reference to **Figure 14**, a flowchart of a process for monitoring accesses to a memory location is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 14** may be implemented in a data cache unit, such as data cache unit **216** in **Figure 2**. This process is used to count accesses to data in a memory location.

At page 37, paragraph 2:

Turning to **Figure 15**, a block diagram illustrating components used for generating meta data, such as performance indicators, is depicted in accordance with a preferred embodiment of the present invention. The compiler supports directives embedded in the source that indicate the meta data to be generated. Compiler **1500** may generate instructions **1502** for execution and meta data for monitoring. As instruction or data cache unit pages are loaded into memory, the

operating system program loader/linker and/or the performance monitoring program, reads the meta data generated by compiler **1500** and loads the meta data into memory, such as performance monitor section **1506**, in these examples. The section itself is marked as meta data **1504.** The processor may accept meta data **1504** in the format of the compiler generated section data in performance monitor section **1506** and populate processor's internal performance instrumentation shadow cache with the data. A block oriented approach is described with reference to **Figure 17** below.

At page 42, last paragraph:

At cache line replacement time, the processor contains new performance instrumentation shadow cache **1730** with cache frames directly associated with the frames in the existing data and instruction caches, such as existing cache **1700**. When the processor's instruction or data cache unit loads a new line, the cache also must load the corresponding perfinst block into the performance instrumentation shadow cache, new performance instrumentation shadow cache **1730**. The processor sees (from the registration data given by the loader at program load time) that the processor is bringing a block into its cache that has an associated perfinst segment, perfinst segment **1728**. The processor looks in translation table **1726** associated with this segment, finds a reference to the perfinst block corresponding to the block it is about to load and loads the perfinst block into new performance instrumentation shadow cache **1730**. In these examples, cache misses associated with meta data are not signaled or are treated differently from cache misses associated data in a primary cache block, such as in primary segment **1702**.

At page 44, last paragraph:

Turning now to **Figure 20**, a flowchart of a process for counting execution for particular instructions is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 20** may be implemented in an instruction cache unit such as instruction cache unit **214** in **Figure 2**.

At page 45, last paragraph:

Turning now to **Figure 21**, a flowchart of a process for counting accesses to a particular memory location is depicted in accordance with a preferred embodiment of the present invention. The

process illustrated in **Figure 21** may be implemented in a data cache unit, such as data cache unit **216** and instruction cache unit **214** in **Figure 2**.

At page 46, paragraph 1:

With reference next to **Figure 22**, a diagram illustrating components used in accessing information collected with respect to the execution of instructions or the access of memory locations in accordance with a preferred embodiment of the present invention. In this example, instruction unit **2200** executes instruction **2202** and increments counter **2204**. This counter is incremented each time instruction **2202** is executed. In this example, instruction unit **2200** may be implemented as instruction cache unit **214** in **Figure 2**.

At page 46, last paragraph:

When the instruction or data cache unit pages are loaded into memory, the operating system program loader/linker and/or the performance monitoring program, reads the meta data generated by the compiler and determines that counting is associated with instruction or data access, then the loading process allocates data areas to maintain the counters as part of its perfinst segment. The size of the counters and the granularity of the data access determine the amount of work area to be allocated.

At page 62, last paragraph:

Data accesses may be monitored in a similar fashion. For example, data **3112** includes data range **3114**. Data accesses to data range **3114** may be counted in a similar fashion to execution of instructions within instruction range **3102** or instruction range **3104**. These ranges may be defined in registers within a data unit, such as data cache unit **216** in **Figure 2**. These ranges for data may be defined in the register as a range of memory locations for the data.

At page 48, paragraph 1:

Data unit **2206** may be implemented as data cache unit ~~**206**~~ **216** in **Figure 2**. In this example, each time data **2208** is accessed, counter **2210** is incremented. Data **2208** and counter **2210** are both located in a particular memory location**.** In these examples, a new instruction may be

employed in which the instruction is called ReadDataAccessCount (RDAC) that takes a data address and a register and puts the count associated with that data address in the register.

At page 58, last paragraph:

Turning next to **Figure 29**, a flowchart of a process for identifying routines that have been executed more than a selected number of times is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 29** may be implemented in a functional unit within a processor, such as instruction cache unit **214** in **Figure 2**. This process is used to identify counts of instructions that are executed and to generate an interrupt when these instructions have occurred more than some selected number of times.

At page 61, paragraph 1:

First, a call stack is examined and the caller of a routine is identified (step **3000**). Next, a count of the number of instructions executed is captured from the instruction cache unit (step **3002**). The count is for a counter used in step **2902** in **Figure 29**. The counter is then reset (step **3004**) with control thereafter returned from the interrupt (step **3006**). The information obtained in the process in **Figure 30** may be used to identify additional routines for monitoring to recursively identify callers of routines.

At page 61, paragraph 2:

Turning next to **Figure 31**, a diagram illustrating ranges of instructions and data that has been selected for monitoring is depicted in accordance with a preferred embodiment of the present invention. In this example, program **3100** includes instruction range **3102** and **3104.** Each of these ranges has been identified as ones of interest for monitoring. Each of these ranges is set within an instruction unit, such as instruction cache unit **214** in **Figure 2**. Each range is used to tell the processor the number of instructions executed in a range, as well as the number of times a range is entered during execution of program **3100**.

At page 61, last paragraph:

Instruction cache unit **3106** uses range registers **3108** to define instruction ranges. These registers may be existing registers or instruction cache unit **3106** may be modified to include

registers to define instruction ranges. These ranges may be based on addresses of instructions. Additionally, range registers **3108** may be updated by various debugger programs and performance tools.

At page 62, paragraph 1:

If an instruction is executed in a range, such as instruction range **3102** or instruction range **3104**, a counter is incremented in instruction cache unit **3106**. Alternatively, the instruction may be sent to a performance monitor unit, such as performance monitor unit **240** in **Figure 2**. The performance monitor unit tracks the count of the number of instructions executed within the range and the number of times the instruction range is entered in these examples.

At page 62, last paragraph:

Turning next to **Figure 32**, a flowchart of a process for counting the number of visits to a set range as well as the number of instructions executed within a set range is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 32** may be implemented in an instruction unit, such as instruction cache unit **214** in **Figure 2**.